

Application note

DA14580 using SUOTA

AN-B-10

Abstract

This document describes how to run a SUOTA demo from iOS and Android platforms, using SmartSnippets.

DA14580 using SUOTA

Contents

Contents	2
Figures.....	3
Tables	3
1 Terms and definitions	4
2 References	4
3 Introduction.....	5
4 Security, external memory and product header parameters	7
4.1 Security when SUOTA is used.....	7
4.2 Choice of the external memory to store the images	8
4.3 Address of the product header.....	8
5 Hardware needed	9
5.1 Central side	9
5.2 Peripheral side	9
5.2.1 Using Dialog's reference designs	9
5.2.2 Using Dialog's PRO Development Kit	9
5.2.3 Using Dialog's BASIC Development Kit	9
6 Running SUOTA with scheme 1.....	10
7 Running SUOTA with scheme 2.....	11
8 Creating the binary files and images & programming the flash.....	12
8.1 Creating the binary files	12
8.2 Creating the image files	13
8.3 Creation of the multi_part.bin for the SPI memory using Scheme 1	14
8.4 Creation of the multi_part.bin for the SPI memory using Scheme 2.....	15
8.5 Preparing the SPI memory: erasing the SPI memory	16
9 Running SUOTA from an iOS platform	18
10 Running SUOTA from an Android platform.....	21
10.1 Running SUOTA with SmartSnippets as a SUOTA Initiator	23
11 Total time vs energy consumption to update a new image	25
11.1 A Real life example	25
11.2 Total time to update a new image.....	26
The total time to update a new image using SUOTA is depending on many parameters:	26
11.2.1 Result in practise	26
11.2.2 Result in theory.....	26
11.3 Energy consumption to update a new image.....	27
12 Revision history	28

DA14580 using SUOTA

Figures

Figure 1: SUOTA feature	5
Figure 2: Security of the service enabled	7
Figure 3: PRO development HW configuration	9
Figure 4: BASIC development HW configuration	9
Figure 5: Memory architecture of scheme 1	10
Figure 6: Memory architecture of Scheme 2	11
Figure 7: Binary & header files created	12
Figure 8: Creation of the first image	13
Figure 9: Creation of the second image	13
Figure 10: The 2 images have been created	13
Figure 11: multi_part.bin file	14
Figure 12: Command line to create the multi_part.bin using Scheme 1	14
Figure 13: multi_part.bin file	15
Figure 14: Command line to create multi_part.bin using Scheme 2	15
Figure 15: Select JTAG connection	16
Figure 16: ERASE operation	16
Figure 17: NON-bootable mode	17
Figure 18: Detection of the DA14580 advertisements	17
Figure 19: Copying images into the SUOTA app	18
Figure 20: First 4 steps to use SUOTA with iOS	19
Figure 21: Second 4 steps to use SUOTA with iOS	19
Figure 22: First four steps when using SUOTA with Android	21
Figure 23 Steps 5 and 6 when using SUOTA with Android	22
Figure 24: SmartSnippets SUOTA Initiator configuration options	23
Figure 25: Real life example	25
Figure 26: Time needed to update a 27 kB image	26

Tables

Table 1: SPOTA profile enabled location in the SW	5
Table 2: Pros and cons of the different schemes	6
Table 3: Different security levels defined in the SW	7
Table 4: Choice of the external memory	8
Table 5: Address of the product header defined in the secondary bootloader project	8
Table 6: Define the MTU size in the SW	25

DA14580 using SUOTA

1 Terms and definitions

BLE	Bluetooth Low Energy (now: Bluetooth Smart)
GAP	Generic Access Profile
GATT	Generic ATTribute Profile
HW	HardWare
MTU	Maximum Transmission Unit
NVDS	Non-Volatile Data Storage
SPOTA	Software Patch Over The Air
SUOTA	Software Update Over The Air
SW	SoftWare

2 References

1. DA14580 datasheet, Dialog Semiconductor
2. UM-B-003, DA14580 Software development guide, User manual, Dialog Semiconductor
3. UM-B-012, DA14580 Creation of a secondary boot loader, User manual, Dialog Semiconductor
4. UM-B-015, DA14580 Software architecture, User manual, Dialog Semiconductor
5. UM-B-018, DA14580 SmartTag reference application, User manual, Dialog Semiconductor
6. UM-B-025, DA14580 Basic Development kit, User manual, Dialog Semiconductor
7. AN-B-003, DA14580 Software Patching over the Air (SPotA), Application note, Dialog Semiconductor
8. AN-B-023, DA14580 Interfacing with external memory, Application note, Dialog Semiconductor
9. UM-B-019, DA14580 Beacon reference design User Manual, Dialog Semiconductor
10. AN-B-001, DA1458x Booting from serial interfaces, Application note, Dialog Semiconductor

DA14580 using SUOTA

3 Introduction

This document describes how to update the image on a DA14580 device that supports Dialog's Software Update Over The Air (SUOTA) proprietary service.

In this document, the following project will be used:

DA14580_581_SDK_3.0.8.0\dk_apps\keil_projects\proximity\prox_reporter.

The DA14580 SDK provides software (for both Central & Peripheral roles) for software update over the air (SUOTA). Over the air update refers simply to a software update that is distributed over a Bluetooth Smart link.



Figure 1: SUOTA feature

This functionality can be achieved by using the SUOTA service. This service is based on Dialog's proprietary Software Patch Over The Air service (SPOTA) which is already implemented in the software stack as shown below. For more info about SUOTA, please see [\[7\]](#).

Table 1: SPOTA profile enabled location in the SW

Parameter	Variable/macro	Source file
SUOTA profile	CFG_PRF_SPOTAR	da14580_config.h

The SPOTAR profile is implemented in the following files:

```
dk_apps\src\ip\ble\h\src\profiles\spotar\spotar\spotar.c
dk_apps\src\ip\ble\h\src\profiles\spotar\spotar\spotar_task.c
```

The SPOTAR application source code is located in the following directory:

```
dk_apps\src\modules\app\src\app_profiles\spotar
```

To run SUOTA, a non-volatile memory (SPI flash or I2C EEPROM) must be hooked up to the DA14580.

SPOTA is instantiated as a GATT Primary Service.

The service exposes a control point to allow a peer device to initiate software update over the air and define two roles:

- The "SPOTA Initiator" which transmits the new software image. It is the GATT client for the SPOTA service (GAP Central Role)
- The "SPOTA Receiver" which receives the new software image, stores the image into the external FLASH/EEPROM device and runs the new image. It is the GATT server for SPOTA service (GAP Peripheral Role).

The proximity reporter (internal processor solution) uses the SPOTA profile to:

DA14580 using SUOTA

- Receive a new software image that is sent by the Central over the Bluetooth Smart link.
- Validate the new image and send informative status updates to the Central.
- Store the new image into an external non-volatile memory (FLASH/EEPROM devices).
- Restart the system.

A dual image bootloader detects and executes the active (latest valid) image. For more information about the secondary bootloader, please refer to [\[3\]](#).

Two different schemes are provided when SUOTA is used:

- **Scheme 1:** The secondary bootloader is stored in the external non-volatile memory.
- **Scheme 2:** The secondary bootloader is burnt into the internal OTP.

The main differences between these 2 schemes are outlined in the table below:

Table 2: Pros and cons of the different schemes

	Scheme 1	Scheme 2
Advantages	OTP can stay blank Useful for development purposes and/or when very low power consumption is not a requirement for the final product.	Fastest boot-up time. Guarantee to boot up anytime.
Disadvantages	In case the external memory is in power down mode and a software reset (e.g.: Watchdog) is triggered, the DA14580 will not boot up properly. The battery has to be removed and replaced.	OTP must be burnt.

4 Security, external memory and product header parameters

4.1 Security when SUOTA is used

The best solution to ensure that an application cannot be hacked or bricked by somebody is to:

- ✓ Use the bonding procedure straight after the connection is established
- ✓ A physical action on the device is needed to delete the bonding information.

The bonding device information is stored in an external non-volatile memory. A push button is used to delete the current bonding information. When no bonding information is stored in the external memory, the device can bond with a new central.

By default, the SUOTA profile uses the Security Mode 1 Level 1: No security level. This can be easily changed in the `app_spotar_enable()` function by adding the following code:

```
#if (BLE_APP_SEC)
    req->sec_lvl = PERM(SVC, UNAUTH); // Security enabled
#else
    req->sec_lvl = PERM(SVC, ENABLE);
#endif // BLE_APP_SEC
```

Figure 2: Security of the service enabled

The security modes and levels which can be used are outlined below:

- ✓ Security Mode 1 Level 1: No security (default mode used when SUOTA profile is activated)
- ✓ Security Mode 1 Level 2: Unauthenticated pairing with encryption
- ✓ Security Mode 1 Level 3: Authenticated pairing with encryption

The table below describes where the security levels can be changed in the software.

Table 3: Different security levels defined in the SW

Parameter	Variable/macro	Source file
Different security levels	gapm_write_att_perm	gapm_task.h

DA14580 using SUOTA

4.2 Choice of the external memory to store the images

The choice of the external memory can be easily selected from the secondary boot loader project as outlined below. By default, a SPI memory type is defined in the `bootloader.h` file.

Table 4: Choice of the external memory

Parameter	Variable/macro	Source file
Use of the SPI Flash	<code>SPI_FLASH_SUPPORTED</code>	<code>bootloader.h</code>
Use of the I2C EEPROM	<code>EEPROM_FLASH_SUPPORTED</code>	<code>bootloader.h</code>

For more information about how to handle external memories, please refer to [\[8\]](#).

4.3 Address of the product header

By default, the address of the product header is at `0x1F000`. In case, the address of the product header has to be changed, two files as outlined below must be changed.

Table 5: Address of the product header defined in the secondary bootloader project

Parameter	Variable/macro	Source file
Address of the product header	<code>PRODUCT_HEADER_POSITION</code>	<code>bootloader.h</code>
Address of the product header	<code>PRODUCT_HEADER_POSITION</code>	<code>app_spotar.h</code>

IMPORTANT NOTE

The product header can be stored at any addresses. By default, it is stored at the address `0x1F000` of the external flash. This can be easily changed as shown below. For more info about the product header, see the section 5 or 6.

DA14580 using SUOTA

5 Hardware needed

5.1 Central side

- The tablets and phones which are running iOS.
- The tablets and phones which are running Android with following versions:
 - Android devices that run version 5.0.0 and above always stall during the image upload.
 - ✓ Android devices that run 4.4.4 work fine.
 - ✓ The SUOTA app is not very stable with devices that run 4.4.2. It works most of the time, but stalling problems during image upload have been seen.

5.2 Peripheral side

5.2.1 Using Dialog's reference designs

- For the Beacon reference design, please refer to [9], section 4.
- For the Proximity Tag reference design, please refer to [5], section 5.

5.2.2 Using Dialog's PRO Development Kit

The picture below shows the right jumper's positions to program the external flash via the JTAG interface. This jumper setting will allow the DA14580 to boot from the external flash too.

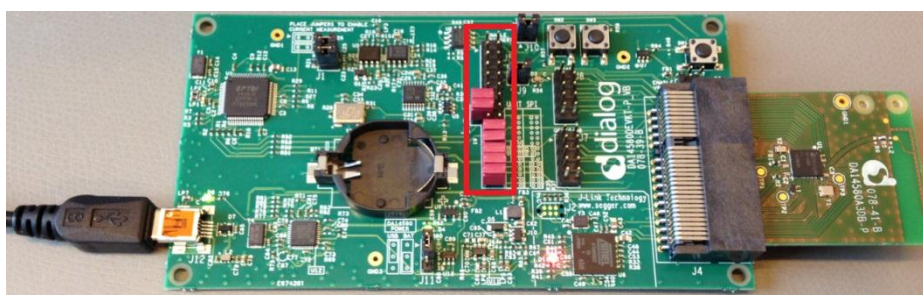


Figure 3: PRO development HW configuration

- For more information about the pro reference design, please refer to: <http://support.dialog-semiconductor.com/resource/pro-all-documents-development-kit-pro-manual-gerber-bom-schematics>

5.2.3 Using Dialog's BASIC Development Kit

The picture below shows the right jumper position to program the external flash via the JTAG interface. This jumper setting will also allow the DA14580 to boot from the external flash.

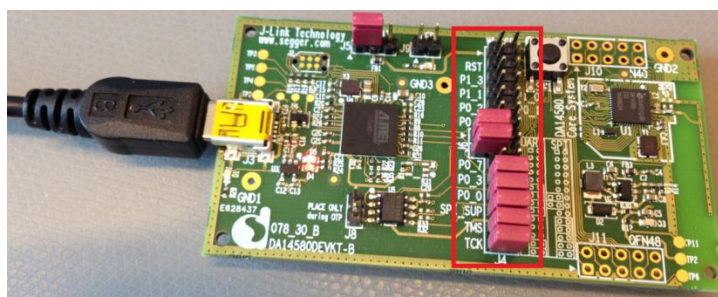


Figure 4: BASIC development HW configuration

- For more information about the basic reference design, please refer to [6].

DA14580 using SUOTA

6 Running SUOTA with scheme 1

The system configuration of scheme 1 is described below:

- SPI/EEPROM flash only (no OTP is used).
- The dual image bootloader is stored at address 0x0 (according to [10]).
- Image #1 is stored at address 0x8000.
- Image #2 is stored at address 0x13000.
- The product header is stored at address 0x1F000.
- Production settings are stored after the product header.

The memory architecture of scheme 1 is shown below:

Address 0x0:

- Ref [10] header

- Dual image bootloader

Address 0x8000:

- Header #1

- Image #1

Address 0x13000:

- Header #2

- Image #2

Address 0x1F000:

- Product header

Ref [10] header

Byte	Field
0	Signature (0x70)
1	Signature (0x50)
2-5	Dummy Bytes
6	Code Size MS Byte
7	Code Size LS Byte
8-	Code Bytes

Product header

Byte	Field
0	Signature (0x70)
1	Signature (0x52)
2	Version MS Byte
3	Version LS Byte
4-7	Offset #1
8-11	Offset #2
12-31	Reserved
32-37	BD Address
38	Reserved
39	XTAL 16 Trim Enable
40-43	XTAL 16 Trim Value
44-63	Reserved
64	NVDS

Same header for the 2 images

Byte	Field
0,1	Signature (0x70, 0x51)
2	imageid
3	validflag
4-7	Code Size
8-11	CRC
12-27	version
28-31	timestamp
32	Encryption flag
33-63	Reserved

Figure 5: Memory architecture of scheme 1

Booting sequence:

- Boot according to [10] from SPI Flash or EEPROM
- The dual image bootloader will:
 - Retrieve the image addresses by reading the product header.
 - Find the last updated (active) image.
 - Load the active image to SRAM and execute the application
- SUOTA for firmware update:
 - Update specific image bank or update an older image

DA14580 using SUOTA

7 Running SUOTA with scheme 2

The system configuration of scheme 2 is shown below:

- SPI/EEPROM flash & OTP are used.
- The dual image bootloader is burnt into the OTP.
- Image #1 is stored at address 0x8000.
- Image #2 is stored at address 0x13000.
- The product header is stored at address 0x1F000.
- Production settings are stored after the product header or in OTP.

The memory architecture of Scheme 1 is shown below:

Address 0x8000:		Same header for the 2 images		Product header	
- Header #1	Header #1	Byte	Field	Byte	Field
		0,1	Signature (0x70, 0x51)	0	Signature (0x70)
		2	imageid	1	Signature (0x52)
- Image #1	Image #1	3	validflag	2	Version MS Byte
		4-7	Code Size	3	Version LS Byte
		8-11	CRC	4-7	Offset #1
Address 0x13000:	Header #2	12-27	version	8-11	Offset #2
		28-31	timestamp	12-31	Reserved
		32	Encryption flag	32-37	BD Address
Address 0x1F000:	Image #2	33-63	Reserved	38	Reserved
				39	XTAL 16 Trim Enable
				40-43	XTAL 16 Trim Value
- Product header	Product header			44-63	Reserved
				64	NVDS

Figure 6: Memory architecture of Scheme 2

Booting sequence:

- System boots in normal mode with a faster booting time than Scheme 1.
- The dual image bootloader will:
 - Retrieve the image addresses by reading the product header.
 - Find the last updated (active) image.
 - Load the active image to SRAM and execute the application
- SUOTA for firmware update:
 - Update specific image bank or update an older image

DA14580 using SUOTA

8 Creating the binary files and images & programming the flash

8.1 Creating the binary files

To create the binary files, the following steps must be carried out.

1. Download DA1458x_SDK_3.0.8.0 zip
2. Open & Build the bootloader project from the path:
tools\secondary_bootloader\secondary_bootloader.uvproj.
3. Copy the secondary_bootloader.hex (from the path
tools/secondary_bootloader/Out) into the mkimage folder (tools/mkimage path).
4. Open proximity reporter project from the path:
dk_apps\keil_projects\proximity\prox_reporter\proxr_reporter.uvproj.
5. Change device name in NVDS (nvds.c) to: NVDS_TAG_DEVICE_NAME = "SUOTA-1"
6. Change the software version in the ble_580_sw_version.h to: DA14580_SW_VERSION
"v_3.0.6.1"
7. Build the project and rename the full_emb_sysram.hex as fw_1.hex.
8. Copy ble_580_sw_version.h (from the path dk_apps\src\dialog\include) into the
mkimage folder (tools/mkimage path) and change the name to fw_version_1.h.
9. Change device name in NVDS (nvds.c) to: NVDS_TAG_DEVICE_NAME = "SUOTA-2"
10. Change the software version in ble_580_sw_version.h to: DA14580_SW_VERSION
"v_3.0.6.2"
11. Build project and copy the generated full_emb_sysram.hex as fw_2.hex
12. Copy ble_580_sw_version.h (from the path dk_apps\src\dialog\include) into the
mkimage folder (tools/mkimage path) and change the name to fw_version_2.h.
13. Convert the 3 .HEX files to .BIN with the hex2bin program (can be found in the
flash_programmer folder). This can be easily done by dragging and dropping the file on the
HexToBin.exe
 - secondary_bootloader.hex -> secondary_bootloader.bin
 - fw_1.hex -> fw_1.bin
 - fw_2.hex -> fw_2.bin

At this point, you should have the following binary & header files added to the mkimage folder:

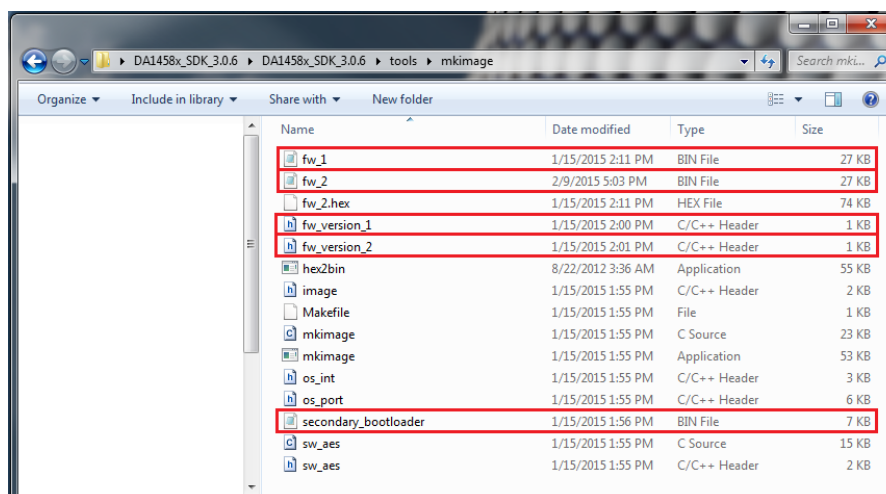


Figure 7: Binary & header files created

DA14580 using SUOTA

8.2 Creating the image files

The images can be created using the mkimage tool from the path `tools\mkimage`. In order to use this tool, the windows command prompt must be open.

IMPORTANT NOTE

The images can be encrypted by using the word 'enc' at the end of the command line. In case the images are encrypted, the secondary bootloader will decrypt them automatically as shown in the `bootloader.c` file.

To create the first image, the command to write is:

```
mkimage.exe single fw_1.bin fw_version_1.h fw_1.img
```

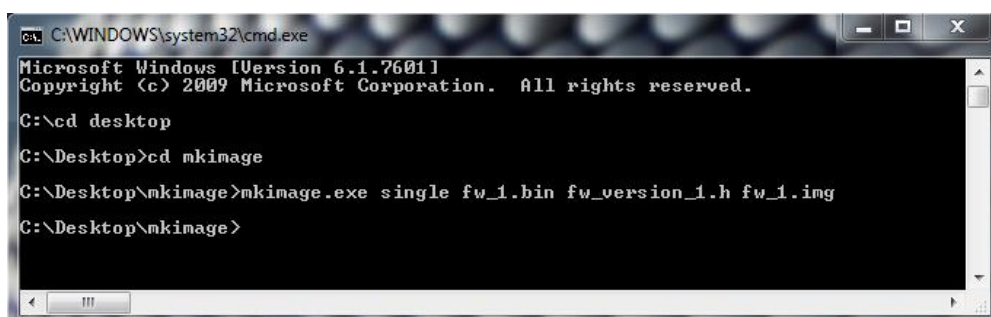


Figure 8: Creation of the first image

To create the second image, the command to write is:

```
mkimage.exe single fw_2.bin fw_version_2.h fw_2.img
```

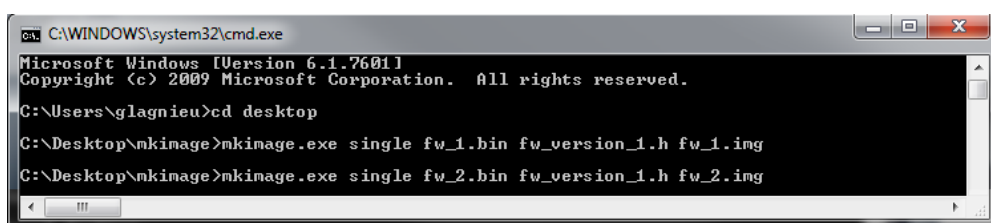


Figure 9: Creation of the second image

At this point, the 2 images have been created as shown below:

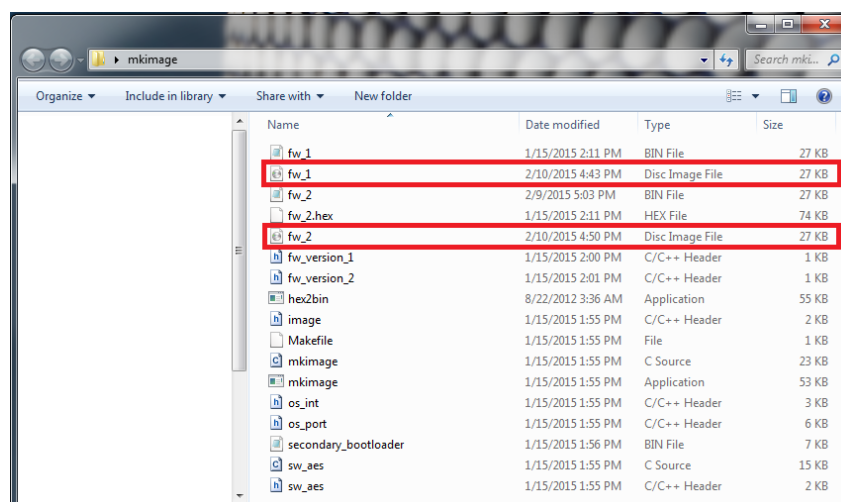


Figure 10: The 2 images have been created

The fact of having 2 images gives the flexibility to have an available image in case the memory gets corrupted in the first image area.

DA14580 using SUOTA

8.3 Creation of the multi_part.bin for the SPI memory using Scheme 1

Scheme 1 has the OTP blank. Therefore, the secondary bootloader will have to be stored in the external memory.

Using Scheme 1, the file `multi_part.bin` which will be programmed into the SPI memory has the outline architecture shown below:

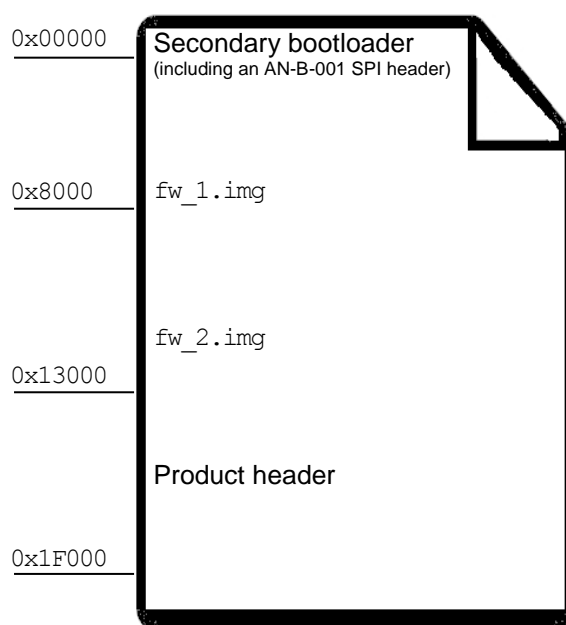


Figure 11: `multi_part.bin` file

To create this binary file, the following command must be used:

```
mkimage.exe multi spi secondary_bootloader.bin fw_1.img 0x8000 fw_2.img
0x13000 0x1F000 multi_part.bin
```

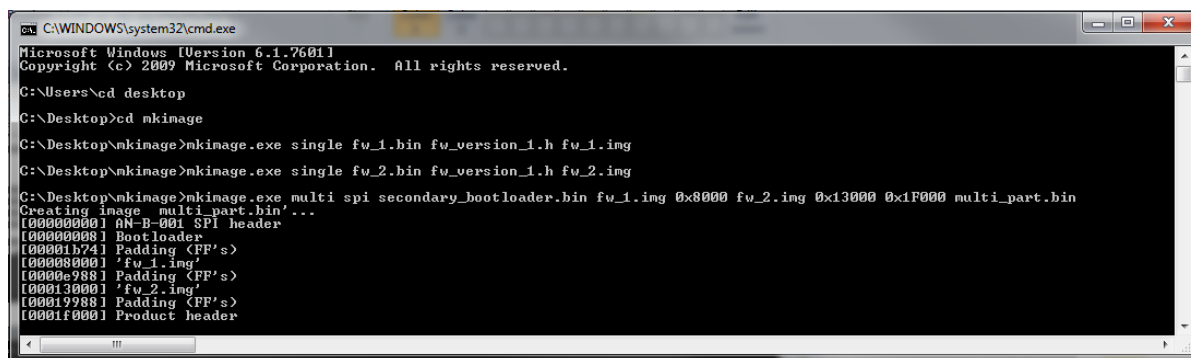


Figure 12: Command line to create the `multi_part.bin` using Scheme 1

Having done that, you can now proceed to section 8.5.

IMPORTANT NOTE

In case, the EEPROM is used as an external memory, the right command is:

```
mkimage.exe multi eeeprom secondary_bootloader.bin fw_1.img 0x8000 fw_2.img
0x13000 0x1F000 multi_part.bin
```

DA14580 using SUOTA

8.4 Creation of the multi_part.bin for the SPI memory using Scheme 2

Scheme 2 has the secondary bootloader burnt in the OTP. Therefore, the secondary bootloader will not be stored in the external memory.

Using Scheme 2, the file `multi_part.bin` which will be programmed into the SPI memory has the outline architecture shown below:

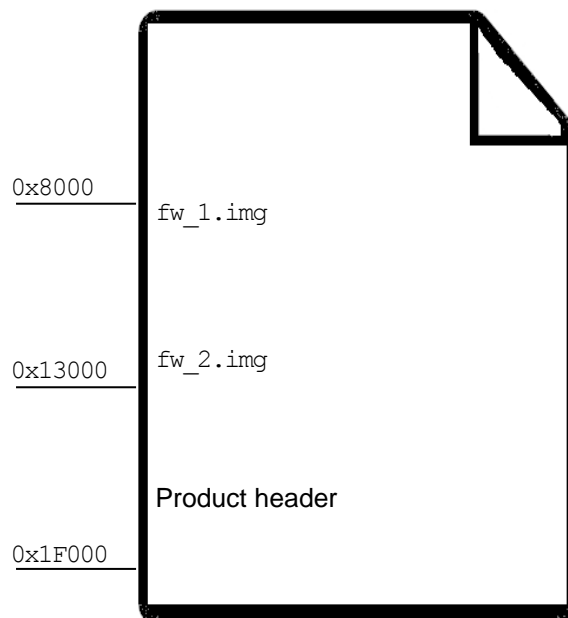
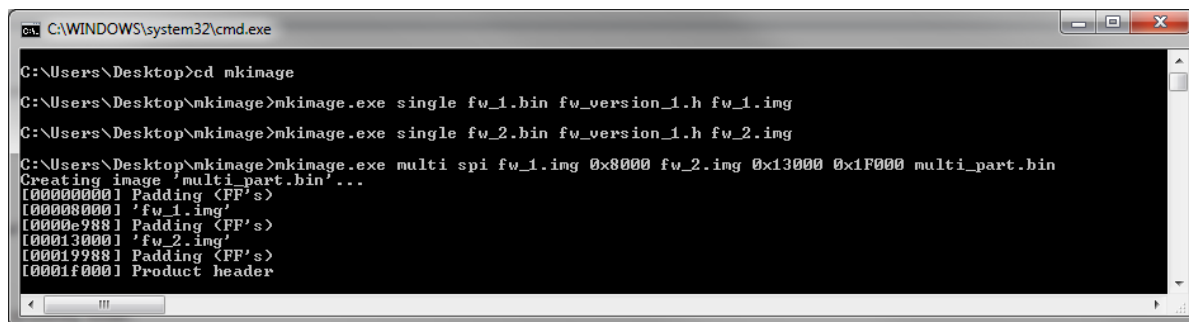


Figure 13: multi_part.bin file

To create this binary file, the following command must be used:

```
mkimage.exe multi spi fw_1.img 0x8000 fw_2.img 0x13000 0x1F000
multi_part.bin
```



```
C:\WINDOWS\system32\cmd.exe
C:\Users\Desktop>cd mkimage
C:\Users\Desktop\mkimage>mkimage.exe single fw_1.bin fw_version_1.h fw_1.img
C:\Users\Desktop\mkimage>mkimage.exe single fw_2.bin fw_version_1.h fw_2.img
C:\Users\Desktop\mkimage>mkimage.exe multi spi fw_1.img 0x8000 fw_2.img 0x13000 0x1F000 multi_part.bin
Creating image 'multi_part.bin'...
[00000000] Padding (FF's)
[00000000] 'fw_1.img'
[0000e988] Padding (FF's)
[00013000] 'fw_2.img'
[00019988] Padding (FF's)
[0001f000] Product header
```

Figure 14: Command line to create multi_part.bin using Scheme 2

Having done that, you can now proceed to section [8.5](#).

IMPORTANT NOTE

If an EEPROM is used as an external memory, the correct command is:

```
mkimage.exe multi eeeprom fw_1.img 0x8000 fw_2.img 0x13000 0x1F000
multi_part.bin
```


DA14580 using SUOTA

8.5 Preparing the SPI memory: erasing the SPI memory

First of all, make sure you have selected the JTAG connection from the SmartSnippet window as shown below:

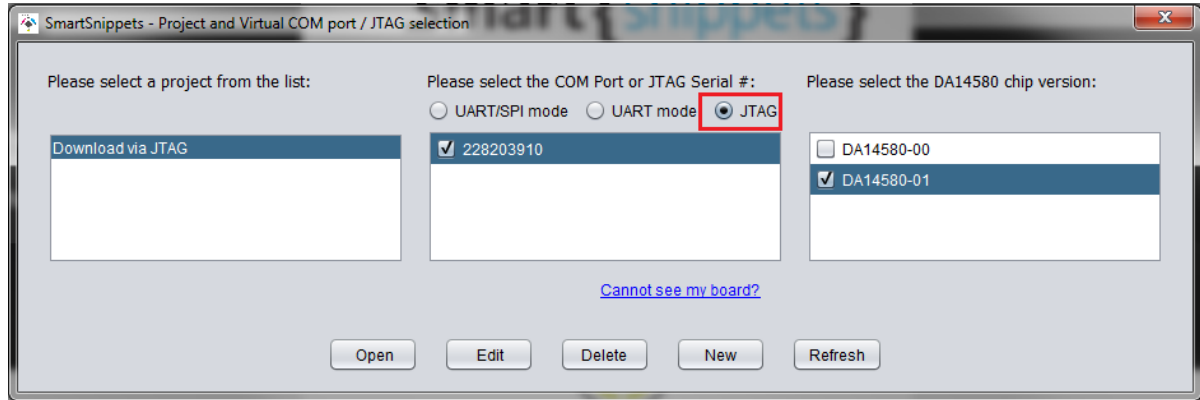


Figure 15: Select JTAG connection

If the external memory is a SPI flash device, proceed as follows:

1. Click on the FLASH tab on the left side of the SmartSnippets windows
2. Select the multi_part.bin file to be downloaded into the external memory
3. Press the 'Connect' button
4. Press the 'ERASE;' button

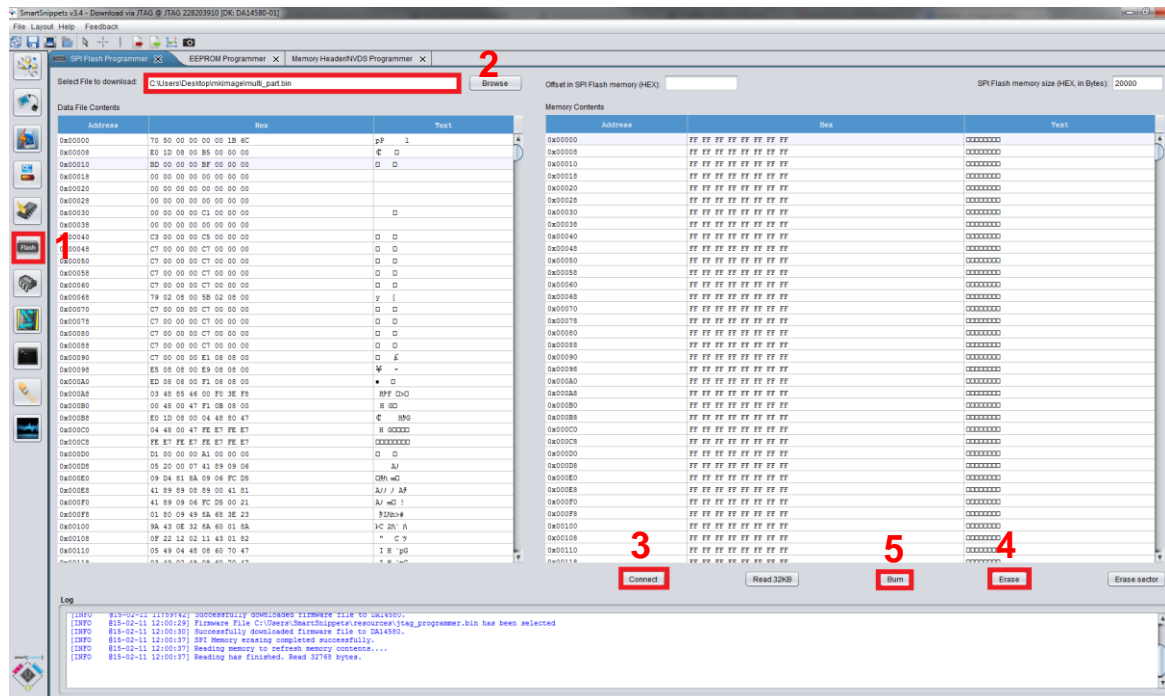


Figure 16: ERASE operation

5. Press the 'Burn' button

After pressing the 'burn' button, the windows shown in Figure 17: NON-bootable mode will appear:

DA14580 using SUOTA

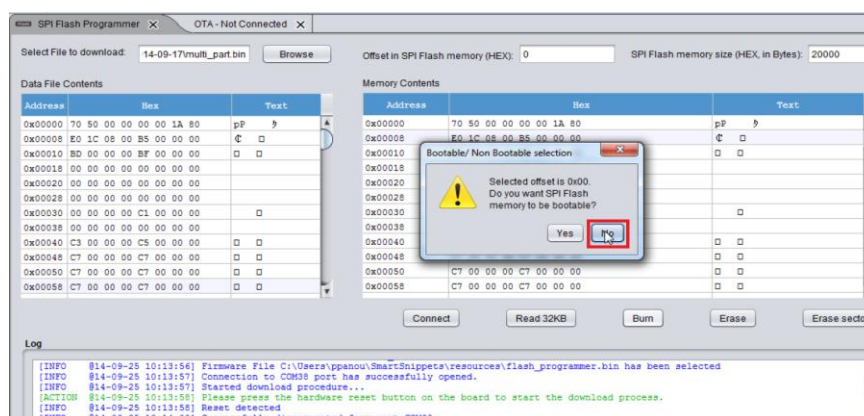


Figure 17: NON-bootable mode

- now press NO in the bootable/Non-bootable pop-up window.

You must now **reset the DevKit** and verify that it has started advertising. The name **SUOTA-1** should be displayed from the SUOTA application (This application is available from both the App store (iOS version) & the Google Play store (Android version)).

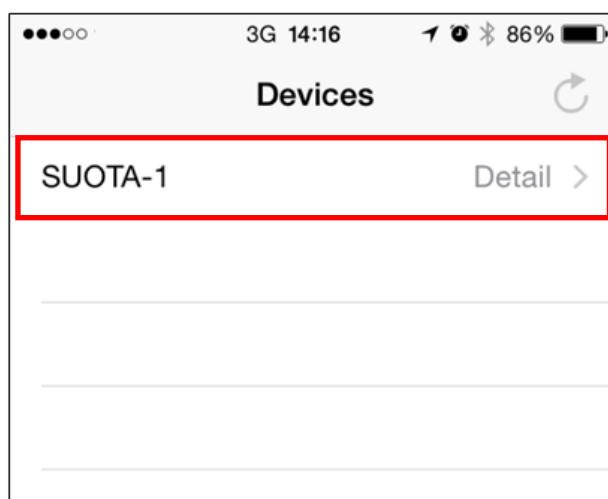


Figure 18: Detection of the DA14580 advertisements

DA14580 using SUOTA

9 Running SUOTA from an iOS platform

IMPORTANT NOTE

If the iOS SUOTA app cannot connect to an advertising DA14580 device make sure any old DA14580 devices that the iOS device has paired in the past are “forgotten” (settings->Bluetooth->click on the “i” next to the device name and select “Forget This Device”).

First, make sure the Dialog SUOTA app is installed (available from the App Store).



Start iTunes, connect the iOS device to the PC via USB and:

1. Go to the ‘Apps’ section
2. Scroll down to ‘File Sharing’ (see 2a, below) and click on SUOTA app (2b)
3. Drag and drop the image files to the ‘SUOTA documents’ section from the mkimage folder.

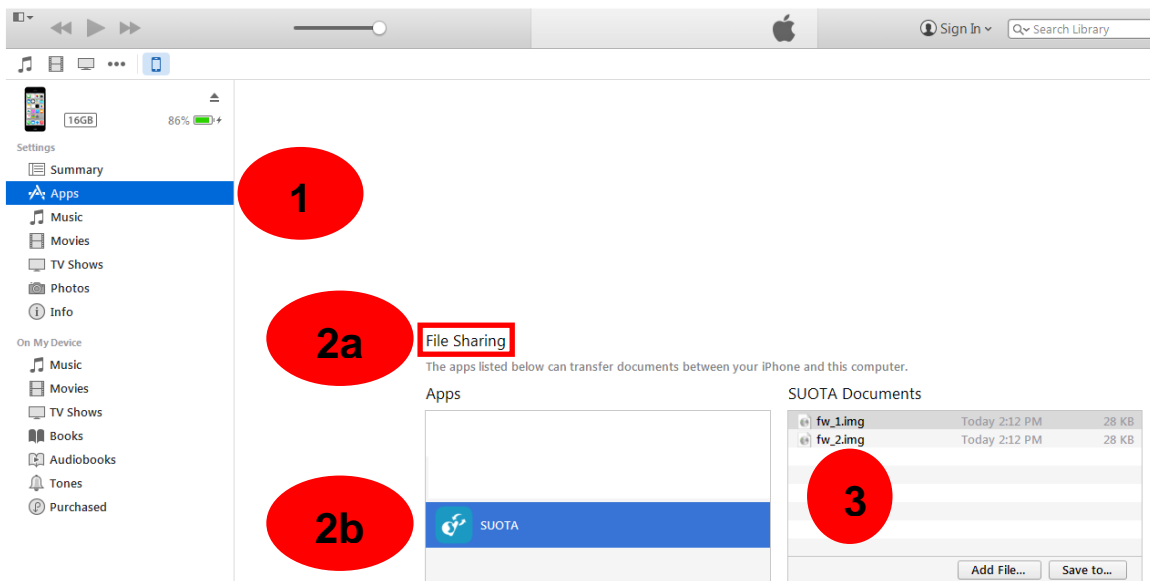


Figure 19: Copying images into the SUOTA app

Then:

1. Start the SUOTA application on the iOS device
2. The DA14580 should advertise at this point and the device name should be detected by the application. If not, click on the clockwise arrow to initiate scanning.
3. Click on the SUOTA-1 device to connect and see the DIS info screen. Verify that the “Firmware rev.” field has the same value as the DA14580_SW_VERSION string set during image creation.
4. After clicking on the “Update” button, the file selection screen appears. Select fw_2.img to update.

DA14580 using SUOTA

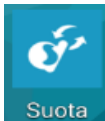
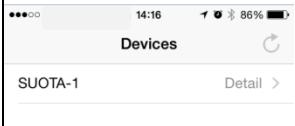
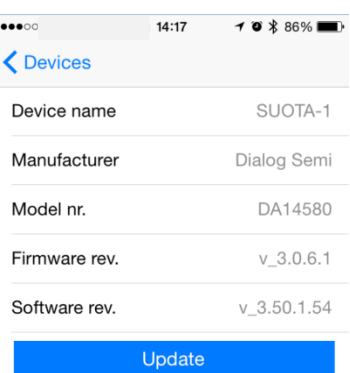
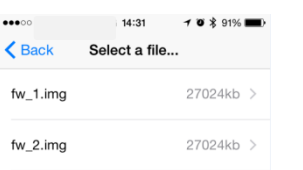
Step 1	Step 2	Step 3	Step 4
			

Figure 20: First 4 steps to use SUOTA with iOS

- After the file selection, the memory parameters configuration screen is shown. In this screen, the default GPIO settings for SPI FLASH configuration are pre-set. Also, the "Image Bank" is set by default to "Oldest" and the "Block size" to "240".
- As soon as the "Send to device" button is pressed, the log screen appears with a status bar.
- When the image is uploaded successfully, reboot the device in order to start advertising as SUOTA-2
- The DA14580 should advertise at this point and the SUOTA-2 device should be detected by the application. Click on the device to connect and verify the "Firmware rev." value.


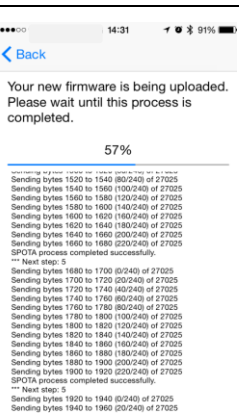
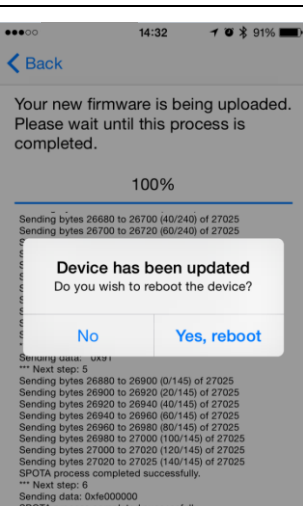
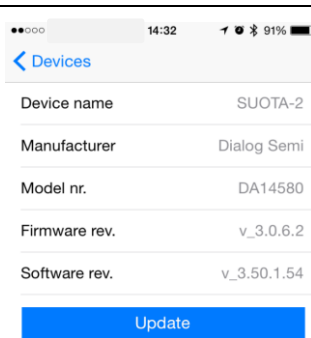
Step 5	Step 6	Step 7	Step 8
			

Figure 21: Second 4 steps to use SUOTA with iOS

IMPORTANT NOTE: AVOID THE SAME IMAGE ERROR

When the user tries to update an image that has the same software version **and** the same timestamp as the new image, a "Same Image Error" message is displayed on the iOS screen.

To avoid this error during a demo do one of the following:

- If two images are used, as in this example, then always update both memory banks with the same image. For example, in this demo description, the SUOTA_1.img was used for both image banks when creating the multi_part.bin (step 11). When the SUOTA app was used to

DA14580 using SUOTA

upload SUOTA_02.img, only one of the memory banks has been updated. The other one still holds SUOTA_1.img. To make sure that the remaining SUOTA_1.img is updated with SUOTA_2.img, upload SUOTA_2.img again. If you want to switch back to SUOTA_1.img, then upload SUOTA_1.img twice to replace both image banks. By uploading the same image twice (replacing the old images in both memory banks), the “Same Image error” is eliminated.

- Create and use three images and sequentially upload one after the other. By doing this it is guaranteed that “Same Image Error” will not happen.
- Note that in normal use the “Same Image Error” rarely happens. Customer will normally create a new image to update an old one. However, in the case of a demo, the same files are used to switch from one image to another and back, so it is possible that a “Same Image Error” might occur if the two memory banks implementation is not well understood.

DA14580 using SUOTA

10 Running SUOTA from an Android platform

IMPORTANT NOTE: Make sure that the SmartTag device is not paired with another device

When the SmartTag device is in Advertising mode, the user can 'forget' a bonded central device by keeping the button pressed for 3 seconds, until a tone is heard. This indicates that the security information has been deleted from the SPI FLASH memory and a new central device can then pair with the SmartTag device.


When deleting the bonding data from the SmartTag SPI FLASH, when it is paired to an Android device, the SmartTag device needs to be removed from the list of paired devices of the Android device (this is usually done via menu *Settings -> Bluetooth -> Forget Device*).

1. Install and start the application on the Android device:

After successful installation, the following icon should appear under the installed applications menu. Click on the icon to start the application.



2. Initial menu, scan for advertising devices:

Press the  icon to initiate scanning. Assuming the SmartTag device is advertising, the device name and the Bluetooth Device address of the device will be displayed as shown in Figure 15.

3. Connect to the SmartTag device:

Click on the SmartTag device to connect. Upon successful connection to the SmartTag device, the DIS information will be displayed on the screen as shown in Figure 22.

4. Update SmartTag software image:

Click on the 'Update device' button and a list of files will appear on the screen. In order for the file to appear in this 'File Selection' screen it has to be copied to the 'Suota' directory of the Android device. **Connect the Android device via USB to the PC where the SmartTag images were created and copy these images under the 'Suota' directory.** An example of the 'File Selection' screen is shown in the table below. Verify that the target image is listed on this screen.

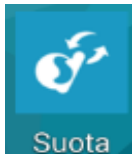
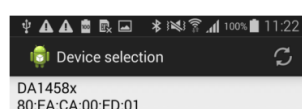

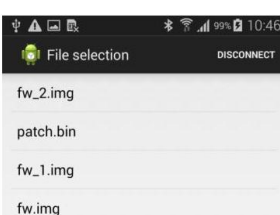
Step 1	Step 2	Step 3	Step 4
			

Figure 22: First four steps when using SUOTA with Android

DA14580 using SUOTA

5. Set the SUOTA parameters:

As soon as the image is selected, the 'Parameter settings' screen appears. See [Figure 23](#) (below) at Step 5.

First, set the memory type. The image update procedure is only supported for non-volatile memory types of SPI (FLASH memory) and I2C (EEPROM). In this example SPI (FLASH) has been selected. Then select the Image (memory) bank:

1: Use the first bank with start address as indicated in the Product Header

2: Use the second bank with start address as indicated in the Product Header

0: Burn the image into the bank that holds the oldest image

Next, define the GPIO pins of the memory device.

In the SmartTag device the SPI FLASH GPIO configuration is as follows:

```
MISO => P0_5
MOSI => P0_6
CS   => P0_3
SCK  => P0_0
```

Finally, scroll down to choose the block size.

A few points should be considered when this size is set:

- Firstly, it has to be larger than 64 bytes, which is the size of the image header.
- Secondly, it must be a multiple of 20 bytes, which is the maximum amount of data that can be written at once in the `SPOTA_PATCH_DATA` characteristic.
- thirdly, it should not be larger than the SRAM buffer in the SUOTA Receiver implementation, which holds the image data received over the BLE link before burning it into the non-volatile memory.

This example uses a block size of 240 bytes. After all the parameters have been set, the user can click on the 'Send to device' button at the bottom of the screen.

6. Reboot the device:

As soon as the 'Send to device' button is clicked, a log screen appears that shows the image data blocks sent over the BLE link. In case an error occurs, a pop up indication will inform the user. When no error occurs and the SmartTag device has received and programmed the image successfully, the screen at Step 6 will appear, prompting the user to reset the SmartTag device.

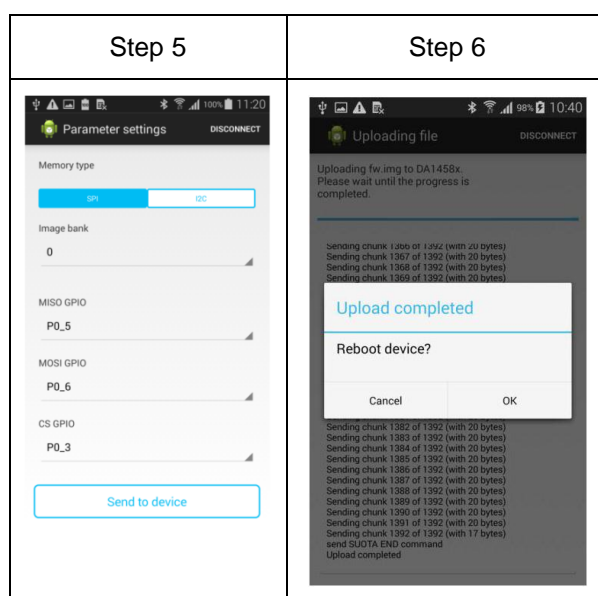


Figure 23 Steps 5 and 6 when using SUOTA with Android

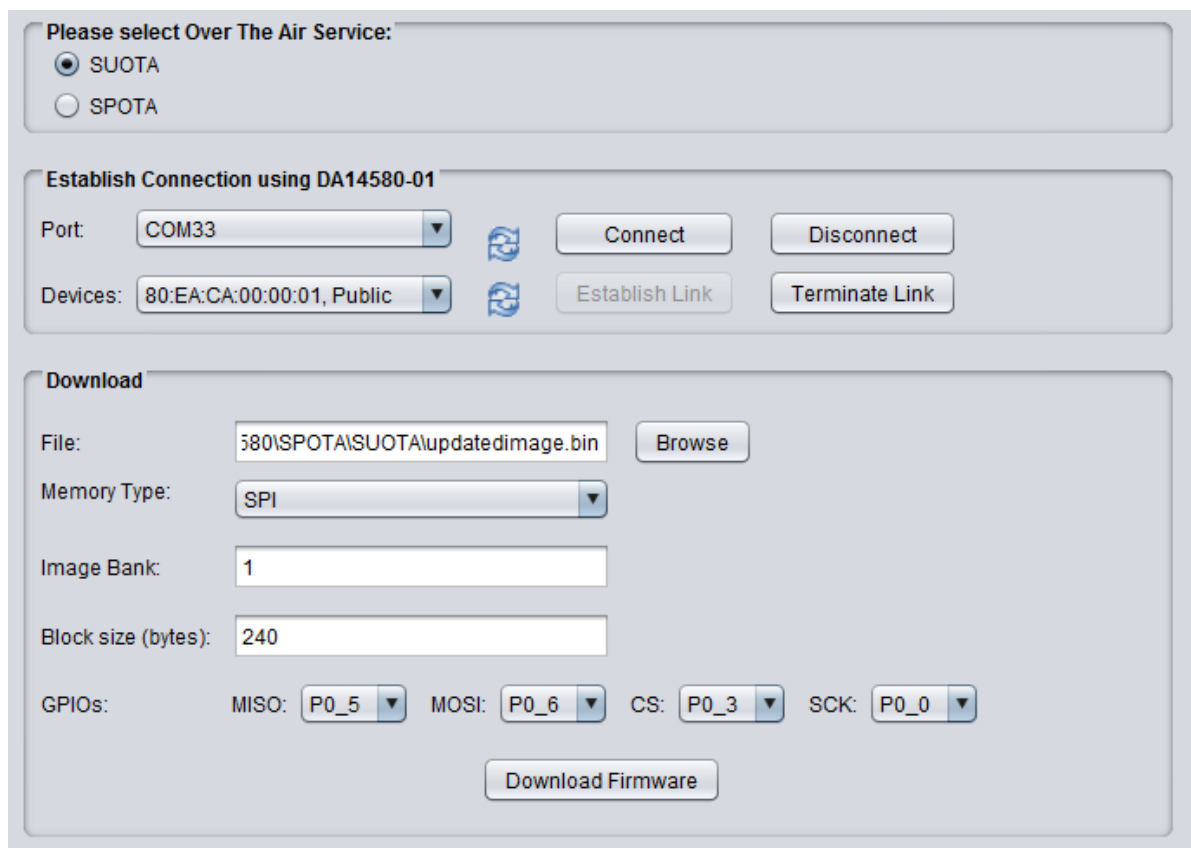
DA14580 using SUOTA

- Verify that the new software is running on the SmartTag device:
Repeat steps 2 and 3 to verify that the DIS screen shows the firmware and software version of the new software.

10.1 Running SUOTA with SmartSnippets as a SUOTA Initiator

SmartSnippets toolkit version 3.2 or later can be used as a SUOTA Initiator. The toolkit's help menu includes a "User Guide" that provides information on how to configure the Initiator to perform an image update. Figure 24 (below) illustrates the SUOTA Initiator screen of the SmartSnippets toolkit. This screen is part of the Over The Air services menu of the toolkit.

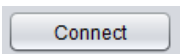
This application, in conjunction with the DA14580 USB Dongle, implements an external processor solution of SUOTA Initiator. Therefore, to run this application it is assumed that the DA14580 USB Dongle is already installed and can be used by the Windows machine.



The screenshot shows the 'Please select Over The Air Service:' section with 'SUOTA' selected. Below is the 'Establish Connection using DA14580-01' section with 'Port' set to 'COM33' and 'Devices' set to '80:EA:CA:00:00:01, Public'. There are 'Connect', 'Disconnect', 'Establish Link', and 'Terminate Link' buttons. The 'Download' section includes a 'File' field with '580\SPOTA\SUOTA\updatedimage.bin', a 'Browse' button, 'Memory Type' set to 'SPI', 'Image Bank' set to '1', 'Block size (bytes)' set to '240', and GPIOs: MISO (P0_5), MOSI (P0_6), CS (P0_3), and SCK (P0_0). A 'Download Firmware' button is at the bottom.

Figure 24: SmartSnippets SUOTA Initiator configuration options

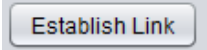
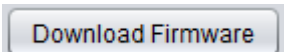
Figure 24 shows an example of how one could configure the Initiator to perform an image update:

- Firstly, choose the Over the Air service (or mode of operation): SUOTA or SPOTA. In this case SUOTA has been selected.
- Select the com port of the DA14580 USB Dongle that is used as the SUOTA Initiator device. When the dongle is inserted in a Windows machine (e.g. laptop) a J-Link device should be discovered in Windows devices and printers. In the J-Link's properties, a JLink CDC UART Port is displayed.
- Press  to download firmware and connect to DA14580 USB dongle. Upon connection, the Initiator will start to scan for advertising SUOTA Receiver devices.

DA14580 using SUOTA

- Assuming that the DA14580 SDK is running the Proximity Reporter application (note that the SUOTA Receiver application is part of the Proximity Reporter Integrated processor project), then the Bluetooth address of the Receiver will be discovered and displayed in :

Devices: 80:EA:CA:00:00:01, Public ▼

- Press  to connect to the Receiver device.
- After the BLE link has been established, browse to find the image file to be sent to the Receiver.
- Choose the memory type. Note that image update is only supported for non-volatile memory types of SPI/FLASH and I2C/EEPROM. In this example SPI has been selected.
- Choose memory bank:
 - “1” means use the 1st bank with start address as indicated in the Product Header
 - “2” means use the 2nd bank with start address as indicated in the Product Header
 - “0” means burn the image into the bank that holds the oldest image
- Choose block size. A few points should be considered when this size is set. Firstly it has to be greater than 64 bytes which is the size of the image header. Secondly, it should be a multiple of 20 bytes which is the maximum amount of data that can be written in SPOTA_PATCH_DATA characteristic at once. Lastly, it should not be greater than the SRAM buffer in the Receiver implementation that stores the image data received over BLE link before it burn them to non-volatile memory. A size of 240 bytes is set in this example.
- Set the GPIO pins of the memory device.
- Press  to initiate the image update process. It is advisable to look at the log window to detect any error messages that will be displayed in red colour.

IMPORTANT NOTE: Block size (in bytes) parameter

The block size can be set higher than 240 bytes. But the larger the block size the larger the RAM buffer required in the SPOTA receiver to store the data before burning the data to flash. The maximum block size can be as big as the RAM buffer you can afford using. Also, it has to be smaller than the size of the image. A size that is a multiple of 20 bytes (the size of the data packet per notification) is advised.

DA14580 using SUOTA

11 Total time vs energy consumption to update a new image

11.1 A Real life example

The environment was as shown below in order to estimate how much time it takes to update the image:

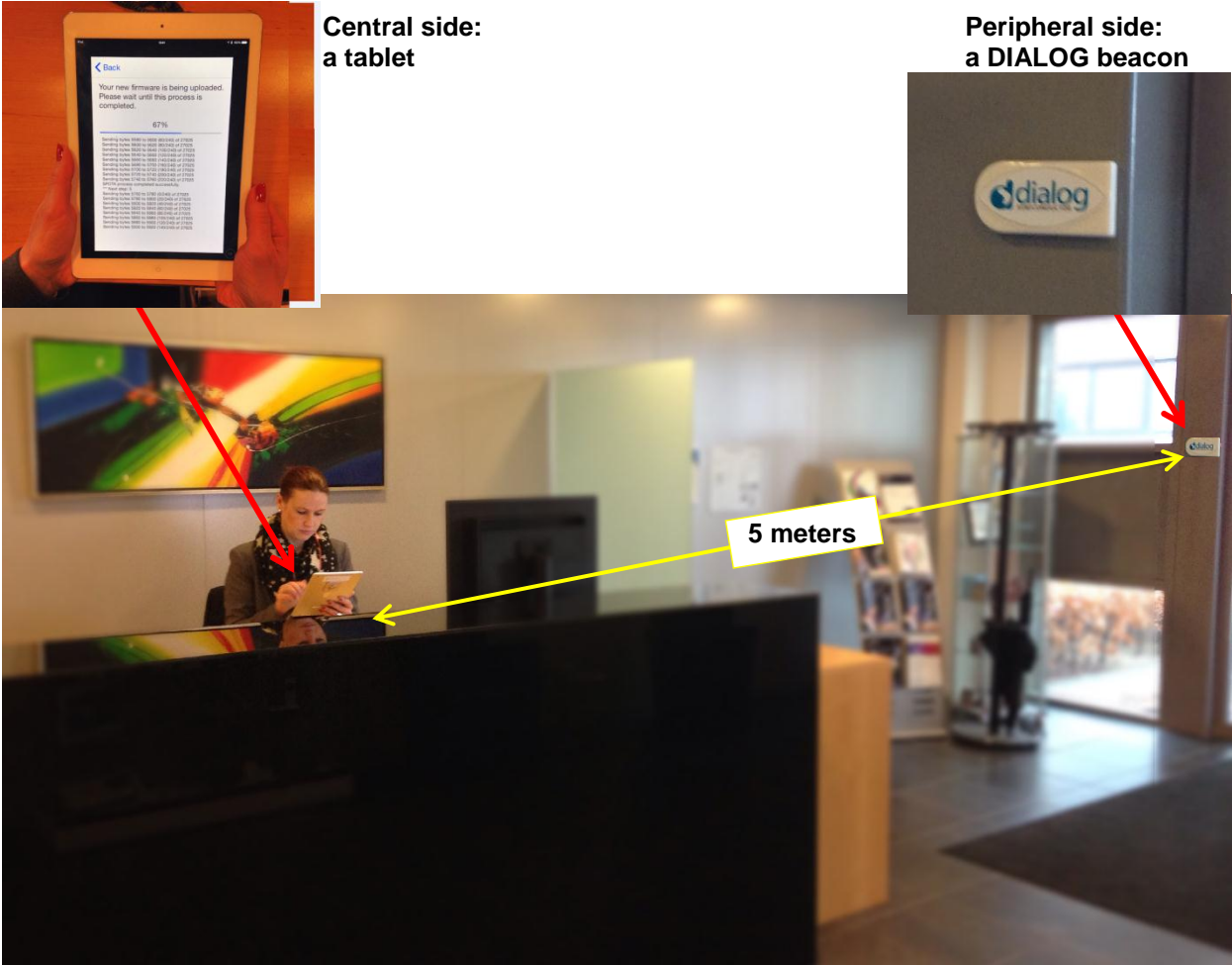


Figure 25: Real life example

- Range from the Peripheral to the Central = 5 meters.
- Average of 4 packets sent per connection interval
- Connection interval = 30 ms (set by the iOS central device)
- Image size = 27 kB
- MTU size = 20 bytes/packet, which is the default size in the software.

Table 6: Define the MTU size in the SW

Parameter	Variable/macro	Source file
MTU size	ATT_DEFAULT_MTU	attn_cfg.h

DA14580 using SUOTA

11.2 Total time to update a new image

The total time to update a new image using SUOTA is depending on many parameters:

- The range from the Central to the Peripheral;
- The environment;
- The average number of packets that the Central can send per connection interval;
- The connection interval set from the Central device;
- The MTU & images sizes.

The following equation gives a roughly estimate of the time needed to update the image:

$$rate \left(\frac{kbit}{s} \right) = 4 (packets) \times 20 \left(\frac{bytes}{packet} \right) \times 8 \left(\frac{bits}{byte} \right) / connection\ interval\ (ms)$$

11.2.1 Result in practise



Figure 26: Time needed to update a 27 kB image

It takes 11.80 seconds to update a new image of 27 kB (including the erasing + writing operation into the flash).

11.2.2 Result in theory

The following formula must be applied:

$$rate \left(\frac{kbit}{s} \right) = 4 (packets) \times 20 \left(\frac{bytes}{packet} \right) \times 8 \left(\frac{bits}{byte} \right) / 30 (connection\ interval\ in\ ms)$$

$$rate \left(\frac{kbit}{s} \right) = 4 \times 20 \times 8 / 30$$

$$rate \left(\frac{kbit}{s} \right) = 21.333\ kbit/sec$$

So, **21 333 bits** are sent **during 1 second**.

The **image size** is 27 kB = 27,648 bytes = **221,184 bits**.

Therefore, the total time needed to update a new image can be calculated as follow:

$$\Delta t = \frac{Image\ size\ (in\ bits)}{number\ of\ bit\ sent\ per\ second} = \frac{221\ 184}{21\ 333} = 10.4\ seconds$$

The total time needed to **update a 27 kB image** is around **10.4 seconds**. Also, the time to erase the flash plus the writing operation time should be added.

DA14580 using SUOTA

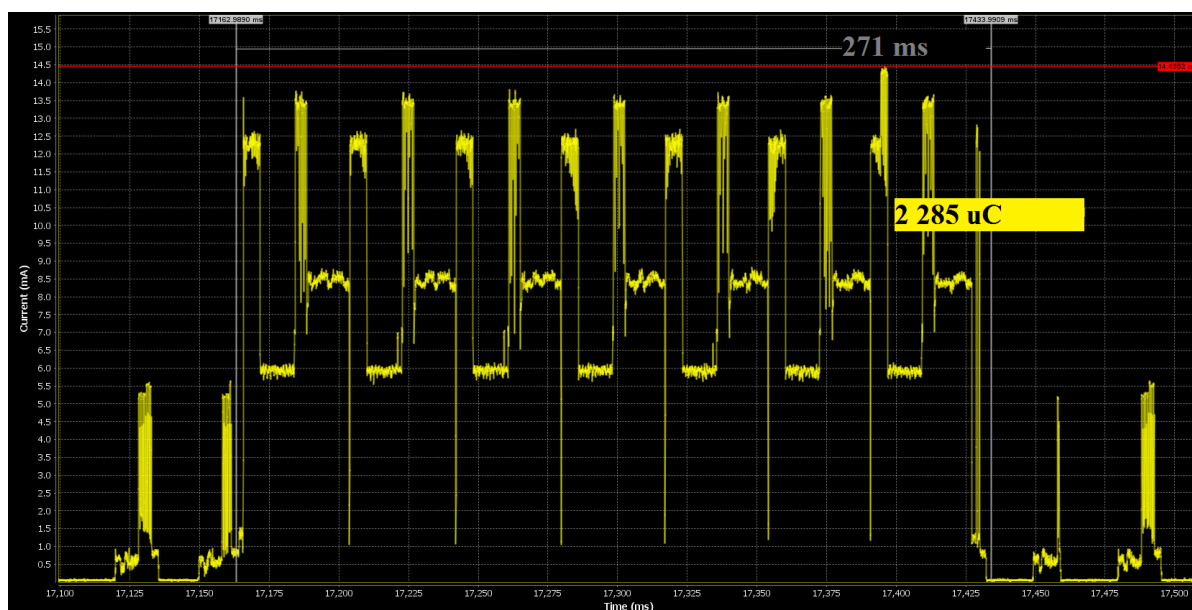
11.3 Energy consumption to update a new image

When SUOTA is running, the image is stored in the external flash memory. Obviously, depending on the external flash memory used, the energy consumption may vary. For more information about external flash/EEPROM, refer to [\[Error! Reference source not found.\]](#).

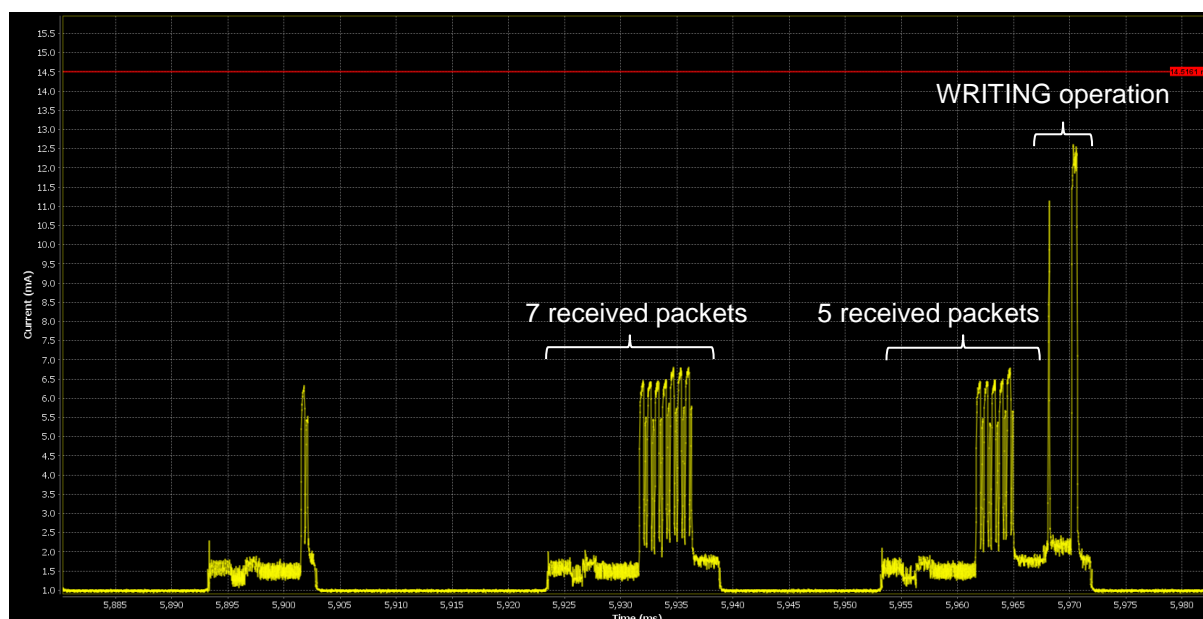
For this example, the MX25V1006E SPI flash memory from MACRONIX was been used. The maximum peak current recorded reached 14.5 mA for 2.4 ms.

When SUOTA is running, the following steps occur:

1. **ERASE the Flash** (time needed: 271 ms, charge: 2 285 μC @3V).



2. **Every 12 packets (240 bytes) sent over the air, a WRITING operation is done.**
(time needed to receive 12 packets + writing operation: 92 ms, charge: 154 μC @3V).



12 Revision history

Revision	Date	Description
1.0	06-July-2015	Initial version.

DA14580 using SUOTA

Status definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

Disclaimer

Information in this document is believed to be accurate and reliable. However, Dialog Semiconductor does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information. Dialog Semiconductor furthermore takes no responsibility whatsoever for the content in this document if provided by any information source outside of Dialog Semiconductor.

Dialog Semiconductor reserves the right to change without notice the information published in this document, including without limitation the specification and the design of the related semiconductor products, software and applications.

Applications, software, and semiconductor products described in this document are for illustrative purposes only. Dialog Semiconductor makes no representation or warranty that such applications, software and semiconductor products will be suitable for the specified use without further testing or modification. Unless otherwise agreed in writing, such testing or modification is the sole responsibility of the customer and Dialog Semiconductor excludes all liability in this respect.

Customer notes that nothing in this document may be construed as a license for customer to use the Dialog Semiconductor products, software and applications referred to in this document. Such license must be separately sought by customer with Dialog Semiconductor.

All use of Dialog Semiconductor products, software and applications referred to in this document are subject to Dialog Semiconductor's [Standard Terms and Conditions of Sale](#), unless otherwise stated.

© Dialog Semiconductor. All rights reserved.

RoHS Compliance

Dialog Semiconductor complies to European Directive 2001/95/EC and from 2 January 2013 onwards to European Directive 2011/65/EU concerning Restriction of Hazardous Substances (RoHS/RoHS2). Dialog Semiconductor's statement on RoHS can be found on the customer portal <https://support.diasemi.com/>. RoHS certificates from our suppliers are available on request.

Contacting Dialog Semiconductor

United Kingdom (Headquarters)

Dialog Semiconductor (UK) Ltd
Phone: +44 1793 757700

Germany

Dialog Semiconductor GmbH
Phone: +49 7021 805-0

The Netherlands

Dialog Semiconductor B.V.
Phone: +31 73 640 8822

Email:

enquiry@diasemi.com

North America

Dialog Semiconductor Inc.
Phone: +1 408 845 8500

Japan

Dialog Semiconductor K. K.
Phone: +81 3 5425 4567

Taiwan

Dialog Semiconductor Taiwan
Phone: +886 281 786 222

Web site:

www.dialog-semiconductor.com

Singapore

Dialog Semiconductor Singapore
Phone: +65 64 849929

China

Dialog Semiconductor China
Phone: +86 21 5424 9058

Korea

Dialog Semiconductor Korea
Phone: +82 2 3469 8200

Hong Kong

Dialog Semiconductor Hong Kong Ltd
Phone: +852 3769 5200